Core Data

- NSEntityDescription:

```
request.entity =
    [NSEntityDescription entityForName:@“Person”
                inManagedObjectContext:context];
```

- Limits:

```
request.fetchBatchSize = 20;
// Anzahl der Objekte, die bei einem Fetch komplett
// im Kontext bereitgestellt werden
request.fetchLimit = 100;
// Anzahl der Objekte, die bei einem Fetch insgesamt
// (ggf. unvollständig) im Kontext bereitgestellt werden
```

Core Data

- NSSortDescriptors
  - einer

```
NSSortDescriptor *sortBy =
    [[NSSortDescriptor alloc] initWithKey:@“name”
                                ascending:YES
                                 selector:@selector(localizedCaseInsensitiveCompare:)];
// ohne selector: @selector(compare:)
```

  - am Request ist eine Liste zu hinterlegen (sortiert wird nacheinander entspr. den Vorgaben, schon in der DB)

```
request.sortDescriptors = [NSArray arrayWithObject:sortBy];
request.sortDescriptors = [NSArray arrayWithObjects:s1,s2,nil];
```

Core Data

- NSPredicate
  - Auswahlkriterium, Strings mit Semantik

```
NSString *someName = @“Olsen”;
NSPredicate *predicate =
[NSPredicate predicateWithFormat:
                    @“name contains %@”, someName];
```

  - diverse Möglichkeiten, z.B.

```
@“uniqueId == %@”, [flickrInfo objectForKey:@“id”]
@“%@ in tags”, (NSManagedObject *)
// tags is a to-many relationship
@“viewed > %@”, (NSDate *)
// viewed is a Date attribute in the data mapping
@"age BETWEEN %@",[NSArray arrayWithObjects:one, ten, nil]
// one, ten: NSNumbers !
@“name contains[c] %@”, (NSString *)
// matches the string in name attribute case insensitively
```

  - und viel viel mehr … s. Predicate Programming Guide

Core Data

- NSPredicate
  - können zusammengesetzt werden

```
NSArray *array =
      [NSArray arrayWithObjects: predicate1, predicate2, nil];

NSPredicate *andPredicate =
    [NSCompoundPredicate andPredicateWithSubpredicates:array];

NSPredicate *orPredicate =
    [NSCompoundPredicate orPredicateWithSubpredicates:array];
```

Core Data

- NSFetchedResultsController
  - verbindet einen FetchRequest mit einem UITableView indem er alle dataSource-Anfragen leicht bedient
  - Erzeugung

```
NSFetchRequest *request = [[NSFetchRequest alloc] init];
// specify request properties ...
NSFetchedResultsController *frc =
  [[NSFetchedResultsController alloc]
      initWithFetchRequest:(NSFetchRequest *)request
      managedObjectContext:(NSManagedObjectContext *)context
        sectionNameKeyPath:(NSString *)sectionKeyOfManagedObjects
        // if nil: one section
                 cacheName:@“SomeCacheName”;
        // if not nil, don’t reuse frc or modify request
[request release];
```

Core Data

- NSFetchedResultsController als dataSource

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [[frc sections] count];
}

- (NSInteger)   tableView:(UITableView *)table
    numberOfRowsInSection:(NSInteger)section {
    id <NSFetchedResultsSectionInfo> sectionInfo =
                    [[frc sections] objectAtIndex:section];
    return [sectionInfo numberOfObjects];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = ...; // get a cell 1. from pool 2. a new one
    NSManagedObject *managedObject = [frc objectAtIndexPath:indexPath];
    // Configure the cell with data from the managed object.
    return cell;
}
```

Core Data

- NSFetchedResultsController als dataSource

```
- (NSString *)    tableView:(UITableView *)tableView
    titleForHeaderInSection:(NSInteger)section {

    id <NSFetchedResultsSectionInfo> sectionInfo =
                        [[frc sections] objectAtIndex:section];
    return [sectionInfo name];
}
```

```
- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView {

    return [frc sectionIndexTitles];
}

- (NSInteger)       tableView:(UITableView *)tableView
  sectionForSectionIndexTitle:(NSString *)title
                      atIndex:(NSInteger)index {

    return [frc sectionForSectionIndexTitle:title atIndex:index];
}
```

Gesture Recognizers

- die Input-Seite von UIViews
  - Verarbeitung von ‚raw touch events' möglich, aber nicht empfohlen
  - stattdessen Gesture Recognizers == Gesten-Erkenner:
    einfach zu implementieren
    einheitliches (look &) feel
- bereitgestellt durch konkrete Ableitungen der (abstrakten*) Klasse `UIGestureRecognizer`

*) by convention only

Gesture Recognizers

- es gibt die folgenden (konkreten) Ableitungen:

```
UITapGestureRecognizer
UIPinchGestureRecognizer
UIRotationGestureRecognizer
UISwipeGestureRecognizer
UIPanGestureRecognizer
UILongPressGestureRecognizer
```

- zur Benutzung sind zwei Dinge erforderlich
  1. ein GR muss einem `UIView` zugeordnet werden
  2. eine Gesten-Behandlungs-Funktion muss implementiert werden

Gesture Recognizers

- 1. wird zumeist im Controller erledigt:
  - View erzeugen, GR erzeugen, dann (z. B. in viewDidLoad)

```
- (void)viewDidLoad {
    UIView *panView = ...;   // dieser soll auf “pan” reagieren
    UIGestureRecognizer *pangr =
        [[UIPanGestureRecognizer alloc]
            initWithTarget:panView action: @selector(pan:)];
    [panView addGestureRecognizer:pangr];
    [pangr release]; // owned now by panView
}
```

- manchmal auch im View selbst (Button, Slider)

Gesture Recognizers

- 2. wird zumeist im View selbst erledigt:

```
- (void)pan:(UIPanGestureRecognizer *)recognizer { ... }
- (void)pan /* for simple tap gesture */          { ... }
```

- aber auch im Controller möglich
- GR's kann man nur an UIViews binden
  - dies kann von jedem Objekt aus geschehen
  - auch die Behandlung kann in jedem Objekt erfolgen

Gesture Recognizers

- in der Behandlungsfunktion kann man die Details der Geste erfragen:
  - Verschiebung/Drehung/Skalierung um …
- Beispiel: `UIPanGestureRecognizer` hat die Methoden

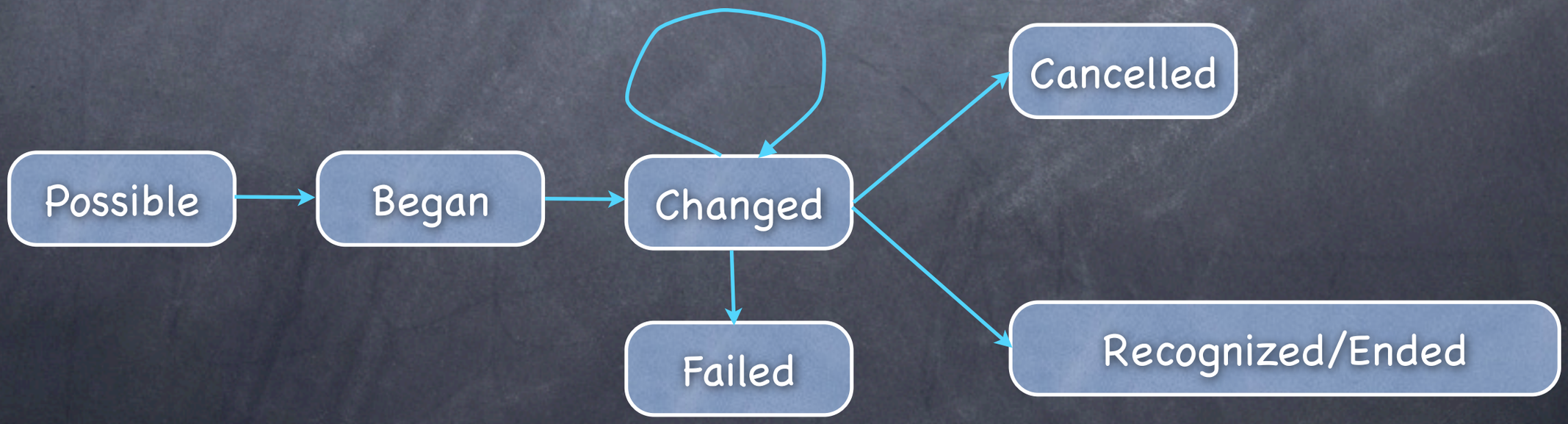
```
- (CGPoint)translationInView:(UIView *)aView;
- (CGPoint)velocityInView:(UIView *)aView;
- (void)setTranslation:(CGPoint)translation
                inView:(UIView *)aView;
// relativ zum View! (self, wenn Handler im View)
```

Gesture Recognizers

```
typedef enum {
   UIGestureRecognizerStatePossible,

   UIGestureRecognizerStateBegan,
   UIGestureRecognizerStateChanged,
   UIGestureRecognizerStateEnded,
   UIGestureRecognizerStateCancelled,

   UIGestureRecognizerStateFailed,

   UIGestureRecognizerStateRecognized = UIGestureRecognizerStateEnded
} UIGestureRecognizerState;
```

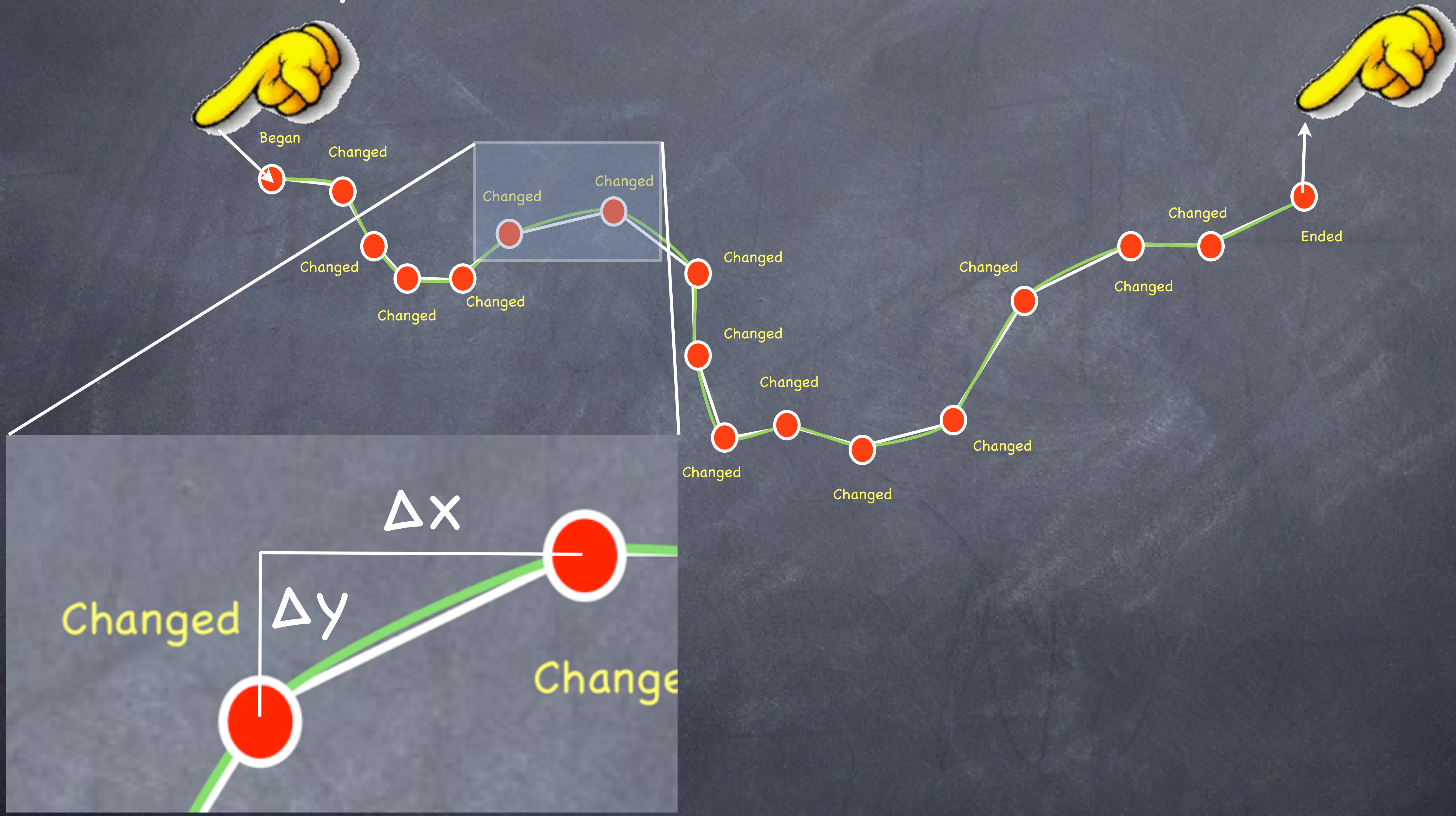
- GR's sind State Machines:
  - haben alle einen

```
@property (readonly) UIGestureRecognizerState state;
```



Gesture Recognizers

- Beispiel: Pan-Gesture



Gesture Recognizers

```
// sample pan implementation in an UIView
- (void)pan:(UIPanGestureRecognizer *)sender {
    if ((sender.state == UIGestureRecognizerStateChanged) ||
        (sender.state == UIGestureRecognizerStateEnded)) {
        CGPoint translation = [sender translationInView:self];
        // move something in myself (I'm a UIView)
        // by translation.x and translation e.g.
        self.origin = CGPointMake(self.origin.x+translation.x,
                                  self.origin.y+translation.y);
        [recognizer setTranslation:CGPointZero inView:self];
        // process last increment only!
        // NO DRAWING CODE HERE !!!
    }
}
```

Gesture Recognizers

- aber der View hat sich doch vermutlich verändert und muss neu gezeichnet werden!?
  - (Custom-) Views so gestalten, dass sie auf Basis von Properties a la `origin`, `scale`, `rotation` von `drawRect` gezeichnet werden.
  - generische Views machen das von Hause aus so :-)
  - dann (getter/setter) geeignet implementieren, z.B.

```
- (void) setOrigin:(CGPoint*) newOrigin {
   // check newOrigin‘s plausibility
   origin = newOrigin;
   [self setNeedsDisplay]; // !!! asap
}
```

Gesture Recognizers

- weitere Gesture Recognizers:
  - UIPinchGestureRecognizer

```
@property CGFloat scale; // reset to 1 in handler

@property (readonly) CGFloat velocity; // how fast
// scale factor per second
```

  - UIRotationGestureRecognizer

```
@property CGFloat rotation; // reset to 0 in handler

@property (readonly) CGFloat velocity; // how fast
// radians per second
```

Gesture Recognizers

- `UISwipeGestureRecognizer`

  eine kontinuierliche Geste, die (nur) diskret erkannt wird

  vorab Konfiguration mittels

  ```
  @property UISwipeGestureRecognizerDirection direction;
  // what direction swipes you want
  @property NSUInteger numberOfTouchesRequired;
  // how many fingers
  ```

  dann auf `Recognized` reagieren

```
typedef enum {
   UISwipeGestureRecognizerDirectionRight = 1 << 0,
   UISwipeGestureRecognizerDirectionLeft  = 1 << 1,
   UISwipeGestureRecognizerDirectionUp    = 1 << 2,
   UISwipeGestureRecognizerDirectionDown  = 1 << 3
} UISwipeGestureRecognizerDirection;
```

- `UITapGestureRecognizer`

  diskret, vorab Konfiguration mittels

  ```
  @property NSUInteger numberOfTapsRequired;
  // single, double, triple tap, etc.
  @property NSUInteger numberOfTouchesRequired;
  // how many fingers
  ```

Gesture Recognizers

- swipe oder pan ?
  - ein View kann durchaus mehrere GR's haben
  - das GR-System kann sie i.allg. gut auseinander halten:
    swipe: schnell und linear
    pan: langsam
  - ausführliches Regelwerk:
    http://developer.apple.com/library/ios/#documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/GestureRecognizers/GestureRecognizers.html

Blocks

- Was ist ein Block?
  - eine „(C-)Funktionsdefinition ohne Namen“ am Ort ihrer Verwendung ≈ C++ lambda

```
[aDictionary enumerateKeysAndObjectsUsingBlock:
    ^(id key, id value, BOOL *stop) {
      NSLog(@“value for key %@ is %@”, key, value);
      if ([@“ENOUGH” isEqualToString:key]) {
        *stop = YES;
      }
}];
```

Blocks

- Was ist ein Block?
  - hat Zugang zu lokalen Variablen seines Kontextes

```
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:
  ^(id key, id value, BOOL *stop) {
      NSLog(@“value for key %@ is %@”, key, value);
      if ([@“ENOUGH” isEqualToString:key] ||
          ([value doubleValue] == stopValue)) {
          *stop = YES;
      }
}];
```

Blocks

- Was ist ein Block?
  - aber nur lesend:

```
BOOL stoppedEarly = NO;
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:
    ^(id key, id value, BOOL *stop) {
        NSLog(@“value for key %@ is %@”, key, value);
        if ([@“ENOUGH” isEqualToString:key] ||
            ([value doubleValue] == stopValue)) {
            *stop = YES;
            stoppedEarly = YES;
        }
}];
```

Assignment of read-only variable 'stoppedEarly'

Blocks

- Was ist ein Block?
  - spezielle Variablen sind auch schreibbar

```
__block BOOL stoppedEarly = NO;
SomeClass *object; // with property ‚early‘
double stopValue = 53.5;
[aDictionary enumerateKeysAndObjectsUsingBlock:
  ^(id key, id value, BOOL *stop) {
     NSLog(@“value for key %@ is %@”, key, value);
     if ([@“ENOUGH” isEqualToString:key] ||
         ([value doubleValue] == stopValue)) {
        *stop = YES; stoppedEarly = YES;
        object.early = YES;
     }
}];
```

Blocks

- Blöcke können als (Pseudo-)Objekte verwendet werden
  - Variablen sind mit der Signatur zu vereinbaren, Syntax a la C function pointers

```
double (^math1) (double) = ^(double x) {return sin(x);}
```

```
typedef double (^MathF) (double);
MathF math2 = ^(double x) {return cos(x);}
```

  - aber nicht mit Funktionszeigern kompatibel

```
int (*one)() = ^{return 1;};
```

Incompatible types in initialization

  - Aufruf aber wie Funktionszeiger

```
NSLog(@"sin(2) = %f", math1(2));
NSLog(@"cos(2) = %f", (math2)(2));
```

Blocks

- Blockobjekte (als Literale) ‚leben' primär auf dem Stack

  The scope of the stack-local data structure is therefore the enclosing compound statement, so you should avoid the patterns shown in the following example(s):

```
void dontDoThis() {
  void (^blockArray[3])(void); // an array of 3 block references
  for (int i = 0; i < 3; ++i) {
    blockArray[i] = ^{ printf("hello, %d\n", i); };
    // WRONG: The block literal scope is the "for" loop
  }
}
```

- können aber zu (länger lebenden) Heap-Objekten kopiert werden, wenn sie außerhalb des Scopes verwendet werden sollen:

```
MathF heapF = [[^(double x){return 0;} copy] autorelease];
```

Blocks

- Blöcke (auf dem Heap) können z.B. in Containern (für einen späteren Aufruf) aufbewahrt werden
- sie unterliegen den üblichen Regeln des Memory Management, verschwinden also mit retaincount == 0
- die einzigen erlaubten Nachrichten an Block-Objekte sind `copy`, `retain`, `release` und `autorelease`
- aus Blöcken referenzierte Objekte werden automatisch retained (inkl. `self` bei Zugriff auf Properties)!
- Blöcke kommen im UIKit häufig vor

```
[UIView animateWithDuration:5.0 animations:^{view.opacity = 0.5; }];
```

Blocks

- Demo block

```
@interface C : NSObject { int i; }
@property int i;
@end

@implementation C
@synthesize i;
@end

int main (int argc, const char * argv[]) {
    NSAutoreleasePool* pool = [[NSAutoreleasePool alloc] init];

    C* c = [[C alloc] init];
    NSLog(@"value of %@ is %d", c, c.i);

    typedef void (^F)();
    F f = ^ {
         c.i = 42;                              // [c retain]; !!!
         NSLog(@"value of %@ is %d", c, c.i);
    }; // in main's scope only

    NSDictionary *dict =
    [NSDictionary dictionaryWithObject: f forKey: @"doIt"];

    [c release]; // NOT GONE, BECAUSE retained by f

    // ([dict objectForKey:@"doIt"])(); // NOT DIRECTLY CALLABLE
    void (^call)() = [dict objectForKey:@"doIt"];
    call();

    [pool drain]; return 0;
}
```